Proceedings of the International Conference
on Computational and Mathematical Methods
in Science and Engineering, CMMSE 2009
30 June, 1–3 July 2009.

Fish Swarm Intelligent Algorithm for Bound Constrained Global Optimization

Edite M. G. P. Fernandes[1], Tiago F. M. C. Martins[2] and Ana Maria A. C. Rocha[1]

[1] *Department of Production and Systems, University of Minho, 4710-057 Braga, Portugal*

[2] *Algorithm R&D Center, University of Minho, 4710-057 Braga, Portugal*

emails: emgpf@dps.uminho.pt, martins.tiago41@gmail.com, arocha@dps.uminho.pt

Abstract

The algorithm herein presented is a modified version of the artificial fish swarm algorithm for global optimization. The new ideas are focused on a set of movements, closely related to the random, the searching and the leaping fish behaviors. An extension to bound constrained problems is also presented. To assess the performance of the new fish swarm intelligent algorithm, a set of seven benchmark problems is used. A sensitivity analysis concerning some of the user defined parameters is presented.

Key words: Global optimization, fish swarm algorithm
MSC 2000: 90C15; 90C56

1 Introduction

In this paper, we consider the problem of finding a global solution of a nonlinear optimization problem with bound constraints in the following form:

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in \Omega, \end{array} \tag{1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is a nonlinear function and $\Omega = \{x \in \mathbb{R}^n : l \leq x \leq u\}$ is the feasible region. The objective function f may possess many local minima in the set Ω since we do not assume that f is convex. Many algorithms have been proposed to solve problem (1), namely those based on swarm intelligence. Probably the most well-known are the particle swarm optimization [2] and the artificial bee colony [1] algorithms. Recently, an artificial life computing algorithm that simulates fish swarm behaviors was proposed and applied in some engineering context with success [3, 4, 5, 6]. The fish swarm behaviors inside water, may be summarized as below:

i) *random* behavior - in general, fish looks at random for food and other companion;

ii) *searching* behavior - when the fish discovers a region with more food, it will go directly and quickly to that region;

iii) *swarming* behavior - when swimming, fish will swarm naturally in order to avoid danger;

iv) *chasing* behavior - when a fish in the swarm discovers food, the others will find the food dangling after it;

v) *leaping* behavior - when fish stagnates in a region, a leap is required to look for food in other regions.

In this paper, we present a new version of the artificial fish swarm algorithm, herein denoted by Fish Swarm Intelligent (FSI) algorithm. Our modifications are focused on:

1. the extension to bound constrained problems meaning that any fish movement will be maintained inside the bounds along the iterative process;
2. modified procedures to translate random, searching and leaping fish behaviors;
3. the introduction of a selective procedure;
4. different termination conditions.

This paper also contains sensitivity studies to clarify the effect of some user defined parameters on the FSI algorithm. The parameters are: i) δ, that defines the size of the visual scope of fish; ii) μ_δ, that defines the reduction factor of δ as iterations proceed; and iii) θ, that gives the fraction of fish that defines the crowded situation inside the visual scope of fish.

The paper is organized as follows. In Section 2 we briefly describe the main fish behaviors that are present in the artificial fish swarm algorithm. Section 3 introduces the new fish swarm intelligent algorithm and gives details concerning the pseudo-codes of the new procedures. Section 4 contains several sets of results obtained when implementing the new FSI algorithm with different values of the user defined parameters, as well as some remarks.

2 The artificial fish swarm paradigm

We will use the words "fish" and "point" interchangeably throughout the paper. The used notation is as follows: $x^i \in \mathbb{R}^n$ denotes the ith point of a population; x^{best} is the point that has the least objective function value and f_{best} is the corresponding function value; $x^i_k \in \mathbb{R}$ is the kth $(k = 1, \ldots, n)$ coordinate of the point x^i of the population; m is the number of points in the population; $nfe_{\max}$ represents the maximum number of function evaluations allowed; ε denotes the precision tolerance; $\delta > 0$ is a fixed visual parameter and $\theta \in (0, 1]$ is a fixed crowded parameter.

The main issue of the artificial fish swarm algorithm is the *visual scope* of each fish. The *visual scope* of each point x^i is defined as the closed neighborhood of x^i with ray equal to a positive quantity called "visual". We use the l_2 norm to define de distance operator. Let Ivisual^i be the set of indices of the points inside the *visual scope* of point x^i, where $i \notin$ Ivisual^i and Ivisual$^i \subset \{1, \ldots, m\}$, and let np^i_{visual} be the number of points in its *visual scope*.

As far as point behaviors are concerned, three possible situations may occur:

- when $np^i_{\text{visual}} = 0$, the *visual scope* is empty, and the point x^i, with no other points in its neighborhood to follow, moves randomly searching for a better region;
- when the *visual scope* is crowded, the point has some difficulty in following any particular point, and searches for a better region choosing randomly another point (from the *visual scope*) and moves towards it;
- when the *visual scope* is not crowded, the point is able either to swarm moving towards the central or to chase moving towards the best point.

The condition that decides when the *visual scope* of x^i is not crowded is

$$\frac{np^i_{\text{visual}}}{m} \leq \theta, \tag{2}$$

where the crowded parameter, θ, is defined as above. In this situation, point x^i has the ability to swarm or to chase. The algorithm simulates both movements and chooses the best in the sense that a better function value is obtained.

The swarming behavior is characterized by a movement towards the central of the points in the *visual scope* of x^i. The central point is then defined by

$$c^i = \frac{\sum_{j \in \text{Ivisual}^i} x^j}{np^i_{\text{visual}}}. \tag{3}$$

The swarming movement is activated only if the central point has a better function value when compared with $f(x^i)$. Otherwise, the point x^i randomly chooses a point inside the *visual scope* and moves towards it if it has a better function value. This is the searching behavior.

The chasing behavior is carried out when the minimum function value inside the *visual scope* of x^i satisfies

$$f(x^{\min}) \equiv \min\left\{f(x^j) : \, j \in \text{Ivisual}^i\right\} < f(x^i) \tag{4}$$

where "min" denotes the index of the point with the least function value. If condition (4) is not satisfied then the algorithm activates the searching behavior. We refer to [6] for some details.

3 Our fish swarm intelligent algorithm

Here we present the general scheme of our FSI algorithm. The main procedures of this algorithm are *Initialize*, *Random*, *Search*, *Swarm*, *Chase*, *Select* and *Leap*, see Algorithm 1. Details concerning these procedures are presented below.

3.1 Initialization

The initial population of m points is randomly generated in the set Ω. Each point x^i in the population is componentwise computed by

$$x_k^i \leftarrow l_k + \lambda(u_k - l_k), \text{ for } k = 1, \ldots, n$$

where λ is assumed to be uniformly distributed between 0 and 1 ($\lambda \sim U(0,1)$). The best and worst function values found in the population are identified:

$$f_{\text{best}} = \min\left\{f(x^i) : i \in \{1, \ldots, m\}\right\} \quad f_{\text{worst}} = \max\left\{f(x^i) : i \in \{1, \ldots, m\}\right\}.$$

3.2 The "visual"

To define the *visual scope*, a fixed "visual" value for all the population depending on the bound constraints of the problem is assumed:

$$\text{"visual"} = \delta \max_{k \in \{1,\ldots,n\}} (u_k - l_k) \tag{5}$$

where δ is the positive visual parameter. In general, this parameter is maintained fixed over the iterative process. However, experiments show that a slow reduction accelerates the convergence to the solution. This will be further discussed in Section 4.

3.3 Random behavior

A point of the population moves randomly when the *visual scope* is empty. Furthermore, when the *visual scope* is crowded and the randomly selected point is not better than x^i, then the random behavior is also activated. Details of our interpretation of a random behavior is shown in the Algorithm 2.

3.4 Swarming, chasing and searching behaviors

The swarming, chasing and searching behaviors can be seen as local behaviors. When the *visual scope* is crowded, see the condition in (2), the algorithm activates the searching behavior and randomly selects a point inside the *visual scope*, i.e., an index, denoted "rand", from the set Ivisual^i is randomly selected and the point x^i is moved towards it if the condition $f(x^{\text{rand}}) < f(x^i)$ holds. The direction of movement is then defined as $d^i = x^{\text{rand}} - x^i$.

When the *visual scope* is not crowded, the algorithm activates two behaviors for x^i. One is the movement towards the central point of the *visual scope*, c^i, computed as

Algorithm 1 fish swarm intelligent algorithm

Input: m, l, u, $nfe_{\max}$, ε, δ, μ_δ, θ, η
iteration $\leftarrow 1$; $\tau \leftarrow 1$
$(x^1, \ldots, x^m) \leftarrow Initialize()$
while termination criteria are not satisfied **do**
  for $i = 1, \ldots, m$ **do**
    Compute the "visual"
    if *visual scope* is empty **then**
      $y^i \leftarrow Random(x^i)$
    else
      if *visual scope* is crowded **then**
        $y^i \leftarrow Search(x^i)$
      else
        if central point is better than x^i **then**
          $y_1^i \leftarrow Swarm(x^i)$
        else
          $y_1^i \leftarrow Search(x^i)$
        end if
        if best function value is better than $f(x^i)$ **then**
          $y_2^i \leftarrow Chase(x^i)$
        else
          $y_2^i \leftarrow Search(x^i)$
        end if
        $y^i \leftarrow \arg\min\{f(y_1^i), f(y_2^i)\}$
      end if
    end if
  end for
  for $i = 1, \ldots, m$ **do**
    $x^i \leftarrow Select(x^i, y^i)$
  end for
  if iteration $> \tau m$ **then**
    if "stagnation" occurs **then**
      Randomly choose a point x^l
      $y^l \leftarrow Leap(x^l)$
    end if
    $\tau \leftarrow \tau + 1$
    $\delta = \mu_\delta \delta$
  end if
  iteration $\leftarrow$ iteration + 1
end while

Algorithm 2 *Random*

input: x^i, l, u, "visual"
for $k = 1, \ldots, n$ **do**
  $\lambda_1 \sim U[0,1]$; $\lambda_2 \sim U[0,1]$
  if $\lambda_1 > 0.5$ **then**
    if $u_k - x_k^i >$ "visual" **then**
      $y_k = x_k^i + \lambda_2$ "visual"
    else
      $y_k = x_k^i + \lambda_2(u_k - x_k^i)$
    end if
  else
    if $x_k^i - l_k >$ "visual" **then**
      $y_k = x_k^i - \lambda_2$ "visual"
    else
      $y_k = x_k^i - \lambda_2(x_k^i - l_k)$
    end if
  end if
end for

shown in (3). Thus, the direction of movement is defined by $d^i = c^i - x^i$ and the new point position is called the trial point y^i. This is the swarming behavior.

The other, called the chasing behavior, considers a movement towards the point that has the least function value, herein denoted by $x^{\min}$. The direction used to compute the new trial point is $d^i = x^{\min} - x^i$.

We recall that both swarming and chasing behaviors are carried out only if c^i, in one case, and $x^{\min}$, in the other, have smaller function values than x^i. Otherwise, a searching behavior is assumed, as described above.

The movement towards any particular point, i.e., along any particular direction, is carried out component by component and takes into account the allowed movement towards the upper bound u_k and lower bound l_k of the set Ω. Furthermore, the direction of movement is normalized so that we can maintain feasibility. Algorithm 3 describes the details of our implementation of a movement along a particular direction d^i.

3.5 Selective procedure

To decide whether or not the previous selected trial point y^i should become the new ith point position, the greedy criterion is used:

$$x^i = \begin{cases} y^i & \text{if } f(y^i) < f(x^i) \\ x^i & \text{otherwise} \end{cases} .$$

Algorithm 3 (Movement along a particular direction)

input: x^i, l, u, d^i
$\lambda \sim U[0,1]$
for $k = 1, \ldots, n$ **do**
  if $d_k^i > 0$ **then**
    $y_k^i \leftarrow x_k^i + \lambda \dfrac{d_k^i}{\|d^i\|}(u_k - x_k^i)$
  else
    $y_k^i \leftarrow x_k^i + \lambda \dfrac{d_k^i}{\|d^i\|}(x_k^i - l_k)$
  end if
end for

3.6 Leaping behavior

When the best objective function value in the population does not change for a certain number of iterations, the algorithm may fall into a local minimum. This is herein denoted by "stagnation". The other points of the population will in the subsequent iterations eventually converge to that local minimum. To be able to leap out the local and try to converge to the global minimum, the algorithm implements a leaping behavior. Thus, when "stagnation" occurs, i.e., when

$$\left| f_{\text{best}}^{(\text{iteration}+m)} - f_{\text{best}}^{(\text{iteration})} \right| \leq \eta$$

holds, for a small positive tolerance η, during every m iterations, the leaping behavior randomly selects a point from the population and carries out a random move inside the set Ω. The Algorithm 4 describes the pseudo-code of this leap procedure. In the algorithm, the notation rand $\sim U\{1, \ldots, m\}$ means a random selection of an integer from the set $\{1, \ldots, m\}$.

Algorithm 4 (Leaping behavior)

input: x, l, u
rand $\sim U\{1, \ldots, m\}$
for $k = 1, \ldots, n$ **do**
  $\lambda_1 \sim U[0,1]$; $\lambda_2 \sim U[0,1]$
  if $\lambda_1 > 0.5$ **then**
    $y_k = x_k^{\text{rand}} + \lambda_2 \left(u_k - x_k^{\text{rand}}\right)$
  else
    $y_k = x_k^{\text{rand}} - \lambda_2 \left(x_k^{\text{rand}} - l_k\right)$
  end if
end for

Table 1: Benchmark problems

	$f(x)$	$[l, u]$
Eason Feton	$0.1(12 + x_1^2 + (1 + x_2^2)/(x_1^2) + (x_1^2 x_2^2 + 100)/(x_1 x_2)^4)$ $f(x^*) = 1.74$	$[0, 10]$
Goldstein Price I	$(1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2))$ $(30 + (20x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$ $f(x^*) = 3$	$[-5, 5]$
Goldstein Price II	$\exp(0.5(x_1^2 + x_2^2 - 25)^2) + (\sin(4x_1 - 3x_2)^4 + 0.5(2x_1 + x_2 - 10)^2$ $f(x^*) = 1$	$[-5, 5]$
Powell Quartic	$(x_1 + 10x_2)^2 + 5(x_3 - x_4)^2 + (x_2 - 2x_3)^4 + 10(x_1 - x_4)^4$ $f(x^*) = 0$	$[-5, 5]$
Rosenbrock	$100(x_2 - x_1^2)^2 + (1 - x_1)^2$ $f(x^*) = 0$	$[-10, 10]$
Six-Hump Camelback	$4x_1^2 - 2.1x_1^4 + (1/3)x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$ $f(x^*) = -1.0316285$	$[-10, 10]$
Wood	$100(x_2 - x_1^2)^2 + (1 - x_1)^2 + 90(x_4 - x_3^2)^2 + (1 - x_3)^2$ $+10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1)$ $f(x^*) = 0$	$[-5, 5]$

3.7 Termination criteria

The algorithm is terminated when one of the following conditions is verified:

$$nfe > nfe_{\max} \text{ or } |f_{\text{worst}} - f_{\text{best}}| < \varepsilon$$

where nfe represents the number of objective function evaluations, $nfe_{\max}$ is the maximum function evaluations allowed and ε is a small positive tolerance.

4 Numerical experiments and remarks

To assess the performance of the herein proposed FSI algorithm, a set of seven well-known benchmark problems is solved. Table 1 describes the objective function, the bounds and the optimum global solution, $f(x^*)$, of each problem. Problems Powell Quartic, Rosenbrock and Wood are unimodal yet difficult to solve. Problems Eason Feton, Goldstein Price I and II and Six-Hump Camelback have several local minima and at least one global minimum. Unless specified otherwise, we run each problem 10 times, each starting from a random population with a different seed. In our study, the number of points in the population depends on n, $m = 10n$. Some of the fixed parameters are as follows: $nfe_{\max} = 250000$, $\varepsilon = 10^{-4}$, $\eta = 10^{-8}$. Here we carry out a sensitivity study for the user defined parameters, δ, μ_δ and θ.

4.1 Study of the visual parameter

To clarify the effect of the visual parameter δ, as well as of its reduction factor μ_δ on the FSI algorithm, we combine three values of δ, namely 0.1, 1 and 10, with two values of the reduction factor μ_δ, 0.9 and 1, and solved the seven problems. Figure 1 shows eight

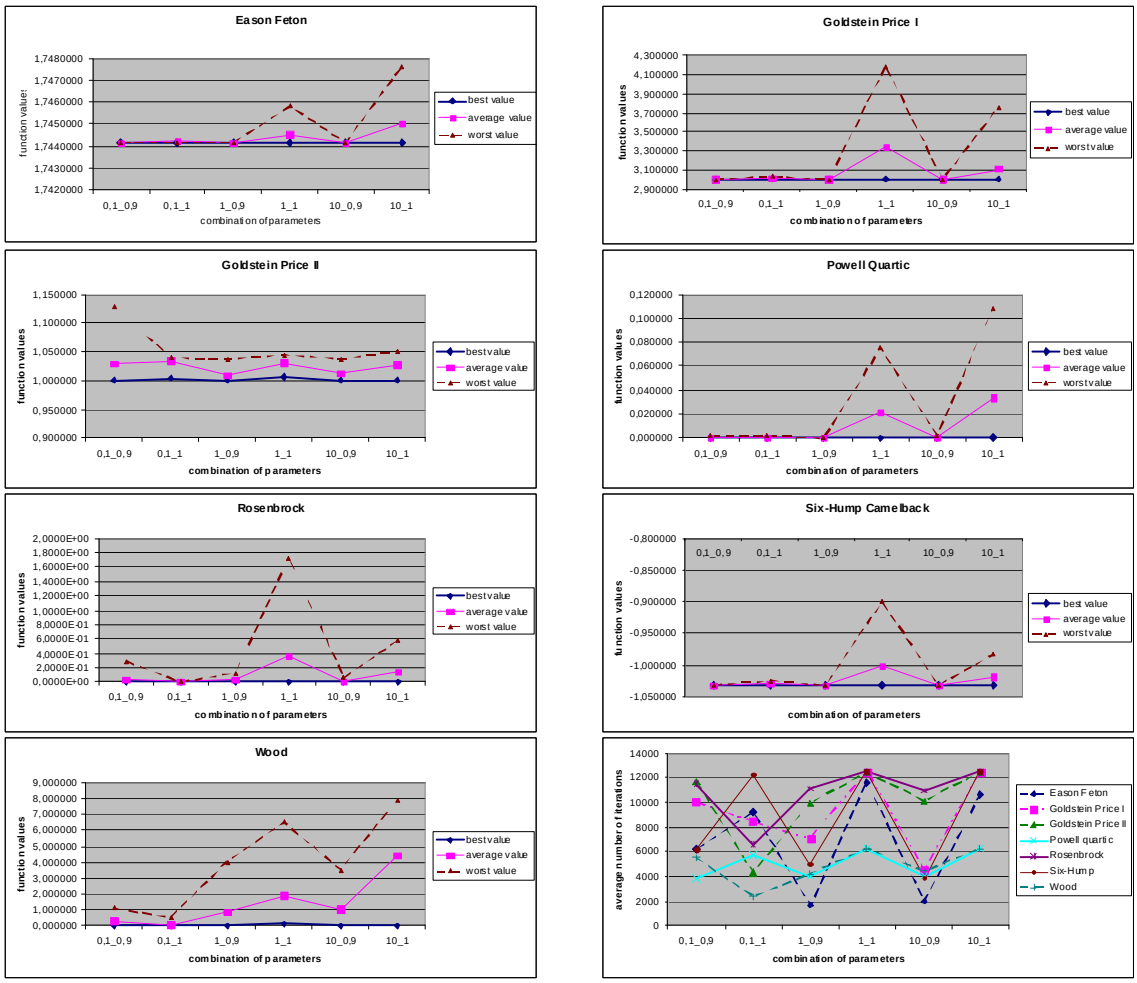

Figure 1: Combination of δ and μ_δ

plots. Each of the first seven illustrates the best function value, the average function value and the worst function value obtained after the 10 runs, for the six cases under analysis: 0.1_0.9 (means that $\delta = 0.1$ and $\mu_\delta = 0.9$), 0.1_1, 1_0.9, 1_1, 10_0.9, 10_1. The frequency for the reduction of the parameter δ is every m iterations (see Algorithm 1). The last plot shows the average number of iterations obtained for each problem and each case. Clearly, the cases $\delta = 1$ with $\mu_\delta = 0.9$ and $\delta = 10$ with $\mu_\delta = 0.9$ lead to excellent solution consistency. In general, they also require less iterations to reach the solution.

4.2 Study of the crowded parameter

Here, we combine three values of δ, namely 0.1, 1 and 10, with two values of θ: 0.8 and 1. The value of μ_δ was maintained fixed as 0.9. Figure 2 presents seven plots with the

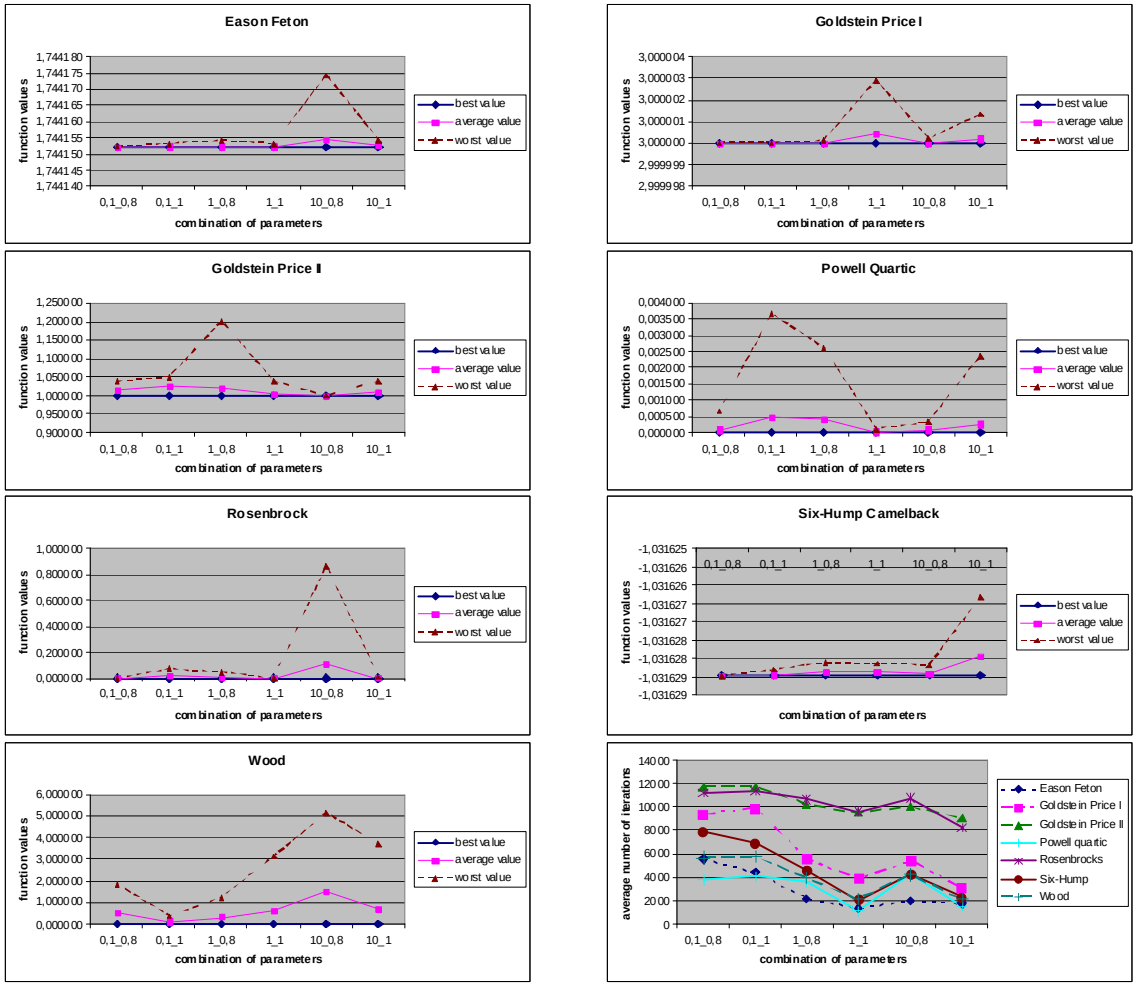

Figure 2: Combination of δ and θ with fixed $\mu_\delta = 0.9$

best function value, the average function value and the worst function value obtained after 10 runs, for the seven problems. We also include a plot with the average number of iterations needed to obtain the required solutions. The results in Figure 2 reveal that the consistency of solution is higher when $\theta = 1$ for some problems and when $\theta = 0.8$ for others. However, we may conclude that a reduction in the number of iterations is obtained in general when $\theta = 1$, with either $\delta = 1$ or $\delta = 10$.

4.3 Other experiments

Here, we aim to analyze for a multimodal and a unimodal problem - Eason Featon and Rosenbrock - the effect of four different values of μ_δ (1, 0.9, 0.75 and 0.5) on the performance of FSI algorithm. During these experiments, $\delta = 1$ and $\theta = 0.8$ were maintained

Table 2: Varying μ_δ using Eason Feton and Rosenbrock problems

μ_δ		Eason Feton	Rosenbrock
1	f_{best}	1.74415459454447	0.00006440796
	st. dev.	6.11E-04	6.14E-01
	nit_{avg}	11614.4	12469.7
0.9	f_{best}	1.74415200564826	0.000411363208
	st. dev.	9.56E-07	3.90E-02
	nit_{avg}	1684.6	11071.6
0.75	f_{best}	1.74415200558774	0.00000419432
	st. dev.	5.39E-08	2.24E-02
	nit_{avg}	11836.5	11890.2
0.5	f_{best}	1.74415200678487	0.00000014653
	st. dev.	3.71E-05	1.60E-01
	nit_{avg}	12269.3	12244.9

Table 3: Different number of runs using Rosenbrock problem

n. runs	f_{best}	f_{avg}	st. dev.	nit_{avg}
10	4.53367E-10	0.006593637	0.017593903	10909.6
30	1.35889E-07	0.011951147	0.038283346	10825.3
60	4.15225E-10	0.022956439	0.063798974	10822.3
100	1.26777E-10	0.017333358	0.046922262	10779.3

fixed. It seems from the results reported in Table 2 that a moderate reduction in the visual parameter (μ_δ set to 0.9 or to 0.75) improves solution consistency, measured by the standard deviation of the function values, "st. dev." in the table, without increasing the computational cost, measured by the average number of iterations, nit_{avg} in the table.

Table 3 lists the results of applying the FSI algorithm to the Rosenbrock problem, using four different values of the parameter number of runs, "n. runs". In these experiments, we used: $\delta = 10$, $\mu_\delta = 0.9$ and $\theta = 0.8$. All runs were stopped with $nfe_{\max} = 250000$. The table reports the f_{best}, the average of the best function values obtained after the specified "n. runs", f_{avg}, "st. dev." and nit_{avg}. The performance of the algorithm does not seem to be affected by the number of runs.

References

[1] D. Karaboga and B. Basturk, *A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm*, Journal of Global Optimization, **39** (2007) 459–471.

[2] J. Kennedy and R.C. Eberhart, *Particle swarm optimization*, IEEE International Conference on Neural Network, (1995) 1942–1948.

[3] M. Jiang, Y. Wang, S. Pfletschinger, M. A. Lagunas and D. Yuan, *Optimal multiuser detection with artificial fish swarm algorithm*, CCIS 2, ICIC 2007, D.-S. Huang, L. Heutte and M. Loog (Eds.), Springer-Verlag, (2007) 1084–1093.

[4] M. Jiang, N. Mastorakis, D. Yuan and M. A. Lagunas, *Image segmentation with improved artificial fish swarm algorithm*, Lecture Notes in Electrical Engineering **28**, Proceedings of the European Computing Conference, N. Mastorakis, V. Mladenov, V.T. Kontargyri(Eds.), ISBN: 978-0-387-84818-1, Springer-Verlag (2009) 133–138.

[5] C.-R. Wang, C.-L. Zhou and J.-W. Ma, *An improved artificial fish-swarm algorithm and its application in feed-forward neural networks*, Proceedings of the Fourth International Conference on Machine Learning and Cybernetics, (2005) 2890–2894.

[6] X. Wang, N. Gao, S. Cai and M. Huang, *An artificial fish swarm algorithm based and ABC supported QoS unicast routing scheme in NGI*, Lecture Notes in Computer Science **4331**, ISPA 2006 G. Min et al.(Eds.), Springer-Verlag, (2006) 205–214.